Bootstrapping Nix and Linux from TinyCC

Linux From tis-but-a-Scratch

Alexander Sosedkin (@t184256)

Red Hat

2023-09-09

What I'll touch up on

- Computing is fun, toy projects are OK
- Side-quests are sometimes more rewarding than main ones
- Bootstrapping is interesting
- Reproducibility is hard
- NixOS is moving towards a bootstrap cooler than mine
- Not all benefits are obvious, not all are even technical

The world needs another Linux distro using Nix!

ZilchOS Core non-goals

- competing with NixOS
- going beyond a Live CD
- systemd
- any software, basically
- flexibility (other than just being small)
- portability
- configurability
- stability
- usability
- practicality

ZilchOS Core goals

- offer just **musl**, **clang**, **busybox**, **Nix** and **Linux**
- target only **one platform**: x86_64 QEMU
- **be lean** enough to experiment on
- **avoid GNU** software where possible
- **force me to learn** more Nix-lang and nixpkgs idioms
- give **content-addressed** Nix a spin
- have a **decent bootstrap** seed/path
- have **fun**

Narrator: That didn't happen.

bootstrap-from-tcc

How to start building a distro

- compiler
- libc
- coreutils
- make
- other stuff

What to compile them with?

NixOS	
24M	`bootstrap-tools.tar.xz` with glibc, gcc and coreutils

What to compile them with?

NixOS

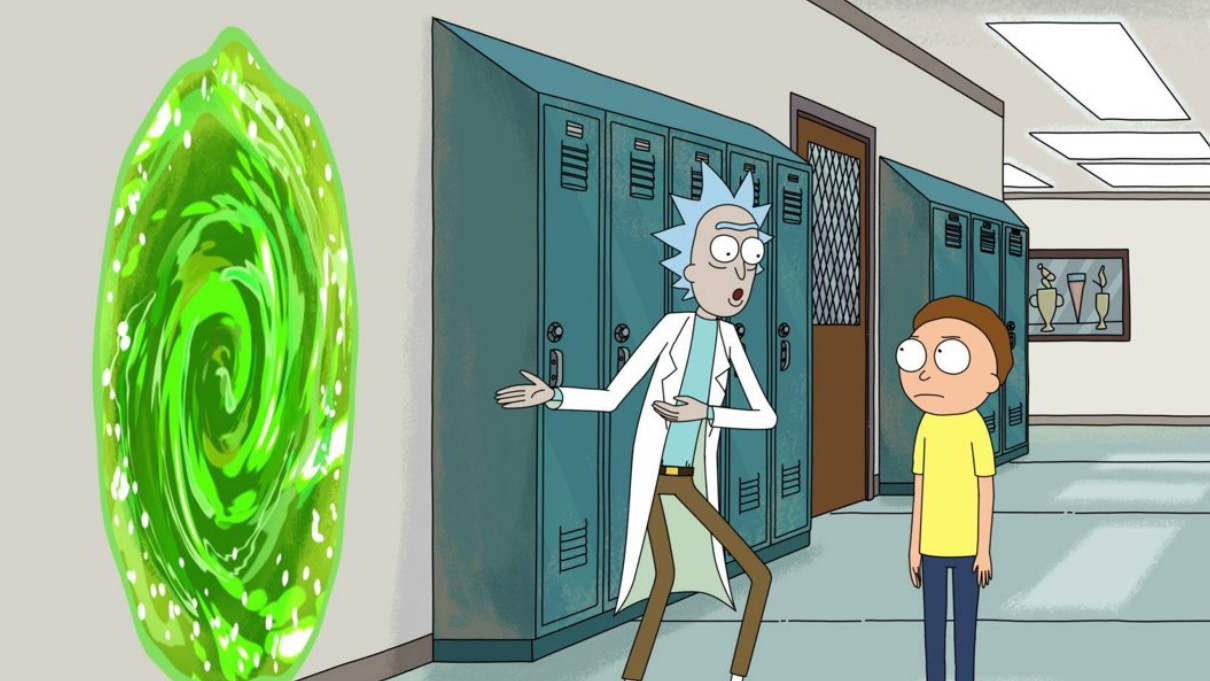
24M bootstrap-tools.tar.xz with glibc, gcc and coreutils

stage0 (bootstrappable.org)

State of art is stage0 and their ‘full-source bootstrap’ from **.5K**

> *This is a set of manually created hex programs*
> *in a Cthulhu Path to madness fashion.*

hex0 -> hex1 -> catm -> hex2 -> M0 -> cc_x86 -> M1 -> M2 -> get_machine -> M2-Planet -> Mes -> tcc -> gcc



TinyCC (tcc)

Another one of Fabrice Bellard's creations.

```
$ nix build 'nixpkgs#pkgsStatic.tinycc' && du -h result/bin/tcc
384K    result/bin/tcc
```

Moreover, it has a `-run` option:

```
  -run          run compiled source
```

```c
#include <stdio.h>
int main() {
    printf("Hello, NixCon!\n");
    return 0;
}
```

Let’s begin from tcc

```c
#include <stdio.h>
int main() {
    printf("Hello, NixCon!\n");
    return 0;
}
```

```
  -nostdinc    do not use standard system include paths
  -nostdlib    do not link with standard crt and libraries
```

-nostdlib: syscalls

```
static long __syscall6(long n, long a1, long a2, long a3, long a4, long a5, long a6);
asm (
        //".globl __syscall6;"
        ".type __syscall6, @function;"
        "__syscall6:;"
        "movq %rdi, %rax;"
        "movq %rsi, %rdi;"
        "movq %rdx, %rsi;"
        "movq %rcx, %rdx;"
        "movq %r8, %r10;"
        "movq %r9, %r8;"
        "movq 8(%rsp),%r9;"
        "syscall;"
        "ret"
);

static __inline long __syscall3(long n, long a1, long a2, long a3) {
        return __syscall6(n, a1, a2, a3, 0, 0, 0);
}
```

-nostdlib: this is my hello world now

```
#define SYS_write 1
#define STDOUT 1

long write(int fd, const void* buf, long cnt) {
        return __syscall3(SYS_write, fd, (long) buf, cnt);
}

int _start() {
        write(STDOUT, "Hello, NixCon!\n",
              strlen("Hello, NixCon!\n"));
        return 0;
}
```

-nostdlib: crawling my way out

libc replacements: `strlen`, `strcpy`, `strcmp`, `memset`, `assert`.

Syscalls: `write`, `open`, `fork`, `execve`, `exit`, `wait4`, `getdents`, `mkdir`.

With unpacked sources and a the possibility to execute itself, let's start compiling towards a shell. `libtcc1`, `protobusybox`, `protomusl`.

Can’t use buildsystems

```
compile_applet("ash",
        PROTOSRC"/protobusybox/shell/shell_common.c",
        PROTOSRC"/protobusybox/shell/ash_ptr_hack.c",
        PROTOSRC"/protobusybox/shell/math.c",
        PROTOSRC"/protobusybox/coreutils/printf.c",
        PROTOSRC"/protobusybox/coreutils/test_ptr_hack.c",
        PROTOSRC"/protobusybox/coreutils/test.c",
        PROTOSRC"/protobusybox/shell/ash.c")
run(42, STORE_PROTOBUSYBOX"/bin/ash", "-c",
        "printf 'Hello from ash!\n'; exit 42");
```

1-stage1.c overview

- (some) tcc
- libtcc1
- protomusl
- tcc
- libtcc1
- tcc (final, independent)
- libtcc1
- protomusl
- tcc (double-check)
- protobusybox

1-stage1.c overview, shortened

- (some) tcc
- protomusl
- tcc
- tcc (final, independent)
- protomusl
- tcc (double-check)
- protobusybox

and, with a shell, we exec into all-past-stage1.sh

2a0-gnumake.sh (shortened)

```
#!/store/1-stage1/protobusybox/bin/ash
export PATH=/store/1-stage1/tinycc/wrappers:/store/1-stage1/protobusybox/bin

mkdir -p /store/2a0-static-gnumake /tmp/2a0-static-gnumake
cd /tmp/2a0-static-gnumake
tar --strip-components=1 -xf /downloads/make-4.4.1.tar.gz
...
ash ./configure \
        --build x86_64-linux \
        --disable-dependency-tracking \
        --prefix=/store/2a0-static-gnumake \
        CONFIG_SHELL='/store/1-stage1/protobusybox/bin/ash' \
        SHELL='/store/1-stage1/protobusybox/bin/ash'
ash ./build.sh
./make -j $NPROC install
```

stage2a: compiler ascension

- gnumake
- binutils
- gnugcc4
- musl
- gnugcc4
- gnugcc10
- linux-headers
- cmake
- python
- clang

stage 2b: rebuild with the new compiler

- musl
- clang
- busybox
- gnumake

So far

```
0 : (some) tcc ->
1 : protomusl -> tcc -> tcc -> protomusl -> protobusybox ->
2a: gnumake -> binutils -> gnugcc4 -> musl -> gnugcc4 ->
    gnugcc10 -> (linux-headers, clang, cmake) -> clang ->
2b: musl -> clang -> busybox -> gnumake
```

Too comfy (2b-gnumake.sh, shortened)

```
#!/store/2b2-busybox/bin/ash
set -uex

export PATH='/store/2b2-busybox/bin'
export PATH="$PATH:/store/2a0-static-gnumake/bin"
export PATH="$PATH:/store/2b1-clang/bin"

mkdir -p /tmp/2b3-gnumake; cd /tmp/2b3-gnumake

tar --strip-components=1 -xf /downloads/make-4.4.1.tar.gz

sed -i 's|/bin/sh|/store/2b2-busybox/bin/ash|' build-aux/install-sh
ash ./configure \
    CONFIG_SHELL=ash SHELL=ash MAKEINFO=true \
    --build x86_64-linux \
    --prefix=/store/2b3-gnumake \
    --disable-dependency-tracking
make -j $NPROC CFLAGS=-O2

./make -j $NPROC SHELL=ash install-strip
```

Let's go Nix!

```
0 : (some) tcc ->
1 : protomusl -> tcc -> tcc -> protomusl -> protobusybox ->
2a: gnumake -> binutils -> gnugcc4 -> musl -> gnugcc4 ->
    gnugcc10 -> (linux-headers, clang, cmake) -> clang ->
2b: musl -> clang -> busybox -> gnumake
3a: sqlite, boost, mbedtls, pkg-config, curl, editline, brotli,
    gnugperf, seccomp, libarchive, libsodium, lowdown
3b: busybox-static, tinycc-static, nix
```

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1
   - filesystem ordering

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1
   - filesystem ordering
2. mismatch in libraries

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1
   - filesystem ordering
2. mismatch in libraries
   - core count non-determinism

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1
   - filesystem ordering
2. mismatch in libraries
   - core count non-determinism
3. race condition in clang buildsystem

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1
   - filesystem ordering
2. mismatch in libraries
   - core count non-determinism
3. race condition in clang buildsystem
   - missing cmake dependency

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1
   - filesystem ordering
2. mismatch in libraries
   - core count non-determinism
3. race condition in clang buildsystem
   - missing cmake dependency
4. mismatch in a single-file tarball!

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1
   - filesystem ordering
2. mismatch in libraries
   - core count non-determinism
3. race condition in clang buildsystem
   - missing cmake dependency
4. mismatch in a single-file tarball!
   - tar 4.15 update

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1
   - filesystem ordering
2. mismatch in libraries
   - core count non-determinism
3. race condition in clang buildsystem
   - missing cmake dependency
4. mismatch in a single-file tarball!
   - tar 4.15 update
5. mismatch in Perl

Intermission: reproducibility

Reproducibility is HARD. It's a constant struggle. Highlights:

1. mismatch in stage1
   - filesystem ordering
2. mismatch in libraries
   - core count non-determinism
3. race condition in clang buildsystem
   - missing cmake dependency
4. mismatch in a single-file tarball!
   - tar 4.15 update
5. mismatch in Perl
   - year change

Wait, there was no Perl

Yes, I've cheated. I'm now building my own fork of Nix that does not depend on openssl.

Other changes in ZilchOS/nix fork include:

- Revive release tarball in a limited form
- Make gtest optional by skipping tests in case of –disable-gtest
- Make libmain explicitly depend on libdl
- Use libsodium for SHA-2 instead of openssl, vendor MD5/SHA1
- Do not fail with inaccessible /proc/self/exe
- Respect NIX_FORCE_BUILD_PATH to set the path w/o sandboxing

Let's restart, this time using Nix

```
0 : (some) tcc ->
1 : protomusl -> tcc -> tcc -> protomusl -> protobusybox ->
2a: gnumake -> binutils -> gnugcc4 -> musl -> gnugcc4 ->
    gnugcc10 -> (linux-headers, clang, cmake) -> clang ->
2b: musl -> clang -> busybox -> gnumake
3a: sqlite, boost, mbedtls, pkg-config, curl, editline, brotli
    gnugperf, seccomp, libarchive, libsodium, lowdown
3b: busybox-static, tinycc-static, nix
4 : protomusl -> tcc -> tcc -> protomusl -> protobusybox ->
    gnumake -> binutils -> gnugcc4 -> musl -> gnugcc4 ->
    gnugcc10 -> (linux-headers, clang, cmake) -> clang ->
    musl -> clang -> busybox
```

What bootstrap-from-tcc is

A toy bootstrap for a toy distro. It just happens to be better than NixOS bootstrap, but worse than stage0. `24M > .5M > .5K`.

It's also not a "trusted" bootstrap, as one has to trust:

- TinyCC binary

What bootstrap-from-tcc is

A toy bootstrap for a toy distro. It just happens to be better than NixOS bootstrap, but worse than stage0. `24M > .5M > .5K`.

It's also not a "trusted" bootstrap, as one has to trust:

- TinyCC binary
- .5G (4.7G) of sources

What bootstrap-from-tcc is

A toy bootstrap for a toy distro. It just happens to be better than NixOS bootstrap, but worse than stage0. 24M > .5M > .5K.

It's also not a "trusted" bootstrap, as one has to trust:

- TinyCC binary
- .5G (4.7G) of sources
- running kernel

What bootstrap-from-tcc is

A toy bootstrap for a toy distro. It just happens to be better than NixOS bootstrap, but worse than stage0. 24M > .5M > .5K.

It's also not a "trusted" bootstrap, as one has to trust:

- TinyCC binary
- .5G (4.7G) of sources
- running kernel

What bootstrap-from-tcc is

A toy bootstrap for a toy distro. It just happens to be better than NixOS bootstrap, but worse than stage0. 24M > .5M > .5K.

It's also not a "trusted" bootstrap, as one has to trust:

- TinyCC binary
- .5G (4.7G) of sources
- running kernel

1. a path to compile Nix from any recent TinyCC without anything
2. same, but with Makefile scaffolding for ease of maintenance
3. an input-less flake that starts with TinyCC and sources and provides a functioning clang musl toolchain

Let's prove it's functioning

In a separate flake, ZilchOS/core, implement `callPackage`/`override` and:

```
4: musl -> clang -> busybox ->
5: (linux-headers, clang, cmake) ->
   clang -> musl -> clang -> busybox ->
   patchelf, ca-bundle, curl, mbedtls,
   boost, brotli, editline, gnugperf,
   libarchive, libsodium, lowdown, nlohmann_json,
   seccomp, sqlite, flex, bison, m4, zstd,
   gnubinutils, gnumtools, gnuxorriso, nasm,
   nix, linux, limine
-> ZilchOS-core-live-cd-2023.09.2.iso
```

Live demo

First release: 2022.02.1 - it boots

Today’s release: 2023.09.2 - self-hosting

What else came out of it

1. a ton of technical experience
2. minimal-bootstrap inspiration
3. patches
4. lessons

Content-addressed derivations

The idea is awesome.
But so far, I regret picking that choice.

A lot of support isn't really there yet:

- passing around nars
- copying derivations between stores
- substituting (for many of the serving software)
- hydra (I have to maintain a patchset)

minimal-bootstrap project inspiration

Emily Trau has packaged stage0-posix in nixpkgs:

> *I've been hacking around reducing nixpkgs's bootstrap binary, and your bootstrap-from-tcc project has been a huge help as a reference! My experiment currently goes from stage0-posix (255 bytes) up to tcc-with-musl.*

Refer to `nixpkgs/pkgs/os-specific/linux/minimal-bootstrap/`.

Join `teams.minimal-bootstrap.members` to help:
artturin, emilytrau, ericson2314, jk, siraben.

Patches

tinycc#da11cf:	don't skip weak symbols during ar…	- accepted
ibara/make#5:	fix compiling with tinycc	- accepted
LLVM D115827:	add missing dependency	- accepted
nix#5678:	document libsodium as a dependency	- accepted
nix#5679:	make cpuid dependency optional	- accepted
nix#5681:	dropping openssl	- rejected
nixpkgs#141999:	fix pkgsStatic.tinycc	- accepted

Takeaways

Technical:

1. Now I know where little distros come from
2. Bootstraping is interesting, even “easy” bootstrapping
3. Reproducibility is hard
4. NixOS is moving towards a stage0-posix bootstrap

Takeaways

Technical:

1. Now I know where little distros come from
2. Bootstraping is interesting, even “easy” bootstrapping
3. Reproducibility is hard
4. NixOS is moving towards a stage0-posix bootstrap

Non-technical:

1. One doesn’t have to do state-of-art stuff
2. Staying a bit beyond ones abilities => constant stream of motivation
3. Side-quests are sometimes more rewarding than main ones

Takeaways

Technical:

1. Now I know where little distros come from
2. Bootstraping is interesting, even “easy” bootstrapping
3. Reproducibility is hard
4. NixOS is moving towards a stage0-posix bootstrap

Non-technical:

1. One doesn’t have to do state-of-art stuff
2. Staying a bit beyond ones abilities => constant stream of motivation
3. Side-quests are sometimes more rewarding than main ones
4. Announce projects without sitting on them for years

Compilers

tcc, tcc, tcc, tcc,
gcc, gcc, gcc,
clang-clang,
clang

tcc, tcc, tcc, tcc
gcc, gcc, gcc,
clang-clang,
clang

clang-clang,
clang

tcc, tcc, tcc, tcc
gcc, gcc, gcc,
clang-clang,
clang

clang-clang,
clang